TITLE:

Design of Conceptual Schemes for the Relational Database Machine Noah

AUTHOR(S):

Yoshikawa, Masatoshi; Le Viet, Chung

CITATION:

Yoshikawa, Masatoshi ...[et al]. Design of Conceptual Schemes for the Relational Database Machine Noah. 数理解析研究所講究録 1983, 495: 1-26

ISSUE DATE:

1983-06

URL:

http://hdl.handle.net/2433/103599

RIGHT:

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

数理解析研究所講究録
495巻 1983年 1-26

Design of Conceptual Schemes for the Relational Database Machine Noah

Masatoshi Yoshikawa
吉川 正俊
(Kyoto University)

Chung Le Viet
レ・ヴィエト・チュン
(HDR Systems Inc.)

1. Introduction

Recently, several commercial RDBMS's (Relational Database Management Systems) and RDBM's (Relational Database Machines) have become available. There are, however, very few database scheme design tools which can be used in practical environment. In this paper, we will discuss the conceptual scheme design problem for the relational database system Noah developed by HDR Systems Inc. [LEVI 8302] [LEVI 83]. The consideration in this paper will provide the logical fundamental for the future implementation of scheme design tools for the Noah system.

The main hardware components of the Noah are the Query Processor and the Intelligent Database Machine (IDM). The IDM was designed and developed by Britton-Lee Inc. [BLI 8109]. The Query Processor has its own micro-processors and system software running on them, and provides the user with a high level interface to the IDM. The system software includes the SQL/Noah parser and the report writer QR.

Utilizing this hardware architecture of the Noah, we will propose the method to enhance the logical functions of the IDM. IDM does not support null values, inclusion dependencies (INDs) or object patterns. The maintenance of these semantic constraints, however, can be achieved by the Query Processor's transformation of data. Thus the Noah seems to provide a high level integrity checking mechanism for the users. We will use the terms "conceptual scheme" and "internal scheme"* to represent the conceptual schemes of the Noah and the IDM, respectively. The Query Processor performs data transformation between the two schemes.

To design these schemes, first the structures of real world data are represented as an enterprise scheme using E-R model. Then it is transformed into conceptual and internal schemes. This design process is summarized in Fig.1.1.

As stated before, in our approach, data structures are first represented using E-R model and then it is transformed into relational conceptual schemes. In Chapter 2, we will state the reason for the adoption of E-R model as a description tool of enterprise schemes in comparison with other design approaches. We have slightly extended Chen's original E-R model to capture the data semantics more precisely. The detail about the extension and an example of enterprise scheme will also be shown. Chapter 3

---

* Note that we use the term "internal scheme" with different meaning from the ordinary usage. It does not mean the physical level detail of DBMSs.

states the transformation from the E-R model into the relational model. The relationship between the conceptual and internal schemes will be given to clarify the underlying idea of the transformation. The handling of null values is a key issue in the transformation. In Chapter 4, the maintenance of several semantic constraints by the Query Processor will be shown. Chapter 5 is the conclusion.

2. The design of enterprise schemes using E-R model

2.1 E-R model as a description tool of real world data

As an initial step of the database scheme design, it is important to decide what exact part of real world data is necessary and to represent them in a comprehensive manner.

There are, in general, the following three approaches for the description of the structure and semantics of real world data in the design of conceptual schemes of relational databases.

(1) use of dependency theory
(2) use of unnormalized relation
(3) transformations from semantically higher data models

Theoretical research on data dependencies has advanced for the last several years, and many dependencies were proposed to represent the semantics of relational schemes. It seems, however, to be difficult for database designers to express the data semantics using data dependencies other than FD. Therefore we consider that approach (1) is not realistic in practical

environment.

Approach (2) is found in [KAMBK8302]. In this approach, at first unformatted sample raw data are analyzed and unnormalized relations are constructed. Starting from these unnormalized relations, designers convert the relational schemes step by step until they finally obtain desirable normalized relations. This approach may especially be suitable for the design of, say research database in bottom up manner. However, we think it is not appropriate to use this approach for the design of large databases of which data structures and semantics are comparatively clear at the initial stage of design process.

In this paper, we will discuss approach (3) and adopt Entity-Relationship model (E-R model) [CHEN 7603] as a semantically higher model. E-R model has the following advantages.

(1) It is popular among other data models.

(2) The data structures of real world are represented comprehensively using Entity-Relationship Diagram (ERD). Therefore the data semantics is easy to understand for designers including non-professional people.

(3) It plays a role for enterprise schemes [CHEN 7706] and is useful also for the user interface [WONGK8209].

As shown in Section 2.2, we will slightly extend Chen's original E-R model to express data semantics more precisely.

2.2 Extended Entity-Relationship Model

In this section we will give the definitions of our E-R model along with each component of ERD. Also relational schemes

corresponding to each component will be given.

(I) Entity Set

In an ERD, an entity set which has $A_1,\ldots,A_n$ as attributes is expressed by a rectangular box with the entity set name in it. Each attribute of an entity set is expressed by an oval, which is linked to the rectangular box of the entity set using an edge. There are several types of edges according to the mapping between an entity set and an attribute. All types of edges are summarized in Table 2.1.

In Table 2.1, "total" means that all entities in the entity set have non-null attribute value(s), while "partial" means that some entities may not have any value for that attribute. We use terms such like "total-m:1-attribute", "partial-attribute", "m:n-attribute", etc. to denote the types of attributes explicitly. If a combination of two or more total-m:1-attribute values has 1:1 mapping with each entity and if any subset of the attribute combination does not have this property, it is called a total key combination. If the combination contains at least one partial-attribute, it is called a partial key combination. All attributes of a total (resp. partial) key combination are encircled by a solid (resp. dashed) line. Among all total-1:1-attributes and total key combinations in an entity set, only one is designated as the entity key. If an entity key is a total-1:1-attribute (resp. total key combination), the edge of the attribute (resp. the circle enclosing the combination) is drawn by a bold line in an ERD.

[Example 2.1]

Let us consider an entity set STUDENT which has attributes S#, YE(year of entrance), SCORE(the score of entrance examination), CAR# and SSN. The ERD of an entity set STUDENT is shown in Fig.2.1. The attribute S# is the entity key and a set of attributes {YE, SCORE} is a partial key combination.

The corresponding relational scheme of this entity set is STUDENT(S#, YE, SCORE, CAR#, SSN), where null values may appear in the attributes SCORE and CAR#. Also FDs: S# -> {YE, SCORE, SSN}, SSN -> S#, {YE, SCORE} -> S# hold in this scheme. Note that the scheme is not in 2NF but in 1NF. A typical object pattern is given in Fig.2.2 where hatched area represents null values.

(II) Relationship Set

A relationship set is a finite set of relationships among entities of distinct entity sets, and may have its own attributes. A relationship set in an ERD is depicted by a diamond-shaped box with the name of relationship set in it. Then that diamond-shaped box is linked with all the entity sets, say $E_1,\ldots,E_m$, relevant to it using edges. These edges may be a solid or dashed one and may have some arrow heads according to the following rules.

(i) An edge between a relationship set R and an entity set E_i $(1\leq i\leq m)$ is solid edge if all entities in E_i are participating in R and dashed edge if some entities in E_i are not participating in R.

(ii) An edge between a relationship set R and an entity set

> E_i ($1 \leq i \leq m$) has an arrow head on the side of E_i iff a particular combination of entities ($e_1,\ldots,e_{i-1}$, $e_{i+1},\ldots,e_m$) appears at most once in R. (If we regard R as a relation scheme with attributes $E_1,\ldots,E_m$, the above definition is equivalent to say that an FD: $E_1,\ldots,E_{i-1}$, $E_{i+1},\ldots,E_m$ ---> E_i holds in R.)

The relationship key of a relationship set R is defined as the set of all the entity keys of E_i ($1 \leq i \leq m$) if no edge between R and E_i's ($1 \leq i \leq m$) has arrow head. Otherwise, if one edge between R and, say E_k has an arrow head, the relationship key becomes the set of the entity keys of E_i ($1 \leq i \leq k-1$, $k+1 \leq i \leq m$). The representation of relationship set's attributes in ERD is same as the case of entity sets.

[Example 2.2]

Consider two entity sets STUDENT, COURSE and a relationship set ATTEND between them. Each ERD in Fig.2.3 is expressing the following data semantics, respectively.

(a) (i) one course has many students as attendees.

(ii) one student can attend to many courses.

(iii) every course has at least one student as an attendee.

(iv) every student attends to at least one course.

(v) some students attending to a course already have a score.

(b) (i), (ii), (iii), (v) in (a) and

(vi) there may be some students not attending to any course.

(c) (i), (iii), (v) in (a), (vi) in (b) and

(vii) one student can attend to at most one course.

The corresponding relational scheme of the relationship set of, say Fig.2.3(b), is ATTEND(S#, C#, SCORE) in which {S#, C#} is the primary key. Attribute SCORE may contain null values and a typical object pattern is shown in Fig.2.4. When transforming a relationship set into a relational scheme, INDs between relational scheme should also be explicitly designated. For example, in the above case, the following two INDs hold.

STUDENT[S#] $\supseteq$ ATTEND[S#]

COURSE[C#] = ATTEND[C#]

(III) Sub-entity Set

Sometimes we need to express an entity set (a sub-entity set) which is a subset of another entity set (a super-entity set) for the following purposes.

(i) to allow entity sets and its sub-entity sets to have different sets of attributes.

(ii) to allow entity sets and its sub-entity sets to have different sets of relationships.

To express a sub-entity set in ERD, we use a wide arrow which emits from an entity set and indicates its sub-entity set as shown in Fig.2.5, where MARRIED_STUDENT is defined as a sub-entity set of an entity set STUDENT. A sub-entity set has the same entity key as its super-entity set. In Fig.2.5, therefore, the entity key of MARRIED_STUDENT is S#. Transforming the ERD into relational scheme, we obtain the two schemes STUDENT(S#, NAME), MARRIED_STUDENT(S#, SP_NAME) and the IND:

STUDENT[S#] ⊇ MARRIED_STUDENT[S#].

In general, one entity set may have many sub-entity sets and each of them, in turn, may have its own sub-entity sets. Thus if all relationship sets were deleted from an ERD, a directed acyclic graph G(V,A) can be obtained. Here V is the set of all entity sets and sub-entity sets, and A is the set of all wide arrows. Each weakly connected component of G(V,A) is called a sub-entity structure. We assume that in each sub-entity structure there is an unique entity set whose indegree is 0. We call this entity set a root entity set. We use the name of the root entity set in a sub-entity structure to represent that sub-entity structure. All entity sets in a sub-entity structure have the same entity key as the root entity set in that sub-entity structure.

[Example 2.3]

Consider two sub-entity structures in Fig.2.6. STUDENT and COURSE are root entity sets in each sub-entity structure. All entity sets in the sub-entity structure STUDENT (i.e. STUDENT, MARRIED_STUDENT, FRESHMAN and MARRIED_FRESHMAN) have S# as entity keys. Also all entity sets in the sub-entity structure COURSE (i.e. COURSE, FUNDAMENTAL_COURSE and ADVANCED_COURSE) have C# as entity keys.

We can regard each sub-entity structure as a generic structure in a similar sense of Smith and Smith's generalization abstraction [SMITS7706].

(IV) Sub-relationship Set

As in the case of sub-entity sets, one relationship set is sometimes a subset of another. For example, consider the ERD shown in Fig.2.7. The relationship set MARKET represents that a part is sold by a supplier. Also BUY represents the fact that a project bought a part from a supplier. It is quite natural to pose a constraint that a project cannot buy a part from a supplier unless the part is sold by the supplier. This constraint can be represented by specifying that BUY is a sub-relationship set of MARKET. In ERD, a wide arrow is used again. The corresponding relational schemes of relationship sets MARKET and BUY are MARKET(P#, S#) and BUY(P#, S#, J#), respectively. The IND: MARKET[P#, S#] $\supseteq$ BUY[P#, S#] holds.

(V) Mutually Exclusive Entity Sets

There are the situations to need to pose the restriction that several entity sets have the same key but are mutually exclusive. For example, let us assume that the employees of an airline company are consisting of pilots and stewardesses. (Here we ignore ground crews etc.) Also assume that no employee can be both a pilot and a stewardess simultaneously. In general, different types of information are need to be stored for each type of occupation, thus those data are better to be managed separately. Fig.2.8 shows the ERD of this example where maneuvable PLANEs are stored for PILOTs and LANGUAGEs they can speak are stored for STEWARDESSes. Similar to the case of sub-entity sets, E# becomes the entity key of PILOT and STEWARDESS.

The corresponding relational schemes are EMPLOYEE(E#, SAL), PILOT(E#, PLANE) and STEWARDESS(E#, LANGUAGE). Also an IND: EMPLOYEE[E#] = PILOT[E#] + STEWARDESS[E#] holds.

Our extension of E-R model here is not based on a formal concept such as "completeness". But we did the extension to make our E-R model powerful enough to be able to describe the semantic constraints which can be maintained on the Noah. Therefore, of course, some semantic constraints cannot be represented using only our E-R model. Furthermore, the equivalence problems of the ERDs under these extension is left open. For the implementation of a useful design tool, these problems must be studied theoretically.

2.3 Enterprise scheme design

As stated in Chapter 1, in our design approach first real world data is represented as an enterprise scheme using E-R model. Since this step is based on trial-and-error, the design tool should support designers by

* notifying designers of violations of design rules,
* improving the schemes automatically, and/or
* prompting designers to input some parameters.

The enterprise scheme which is finally obtained in this step could also be used as a user interface [WONGK8209].

As an example, assume that we have obtained an enterprise scheme of which ERD is shown in Fig.2.9. The semantics of the ERD will be self-explanatory. We will use this scheme as a running

example in the following chapters.

3. The design of conceptual schemes

After designing an enterprise scheme, designers need to transform it into a conceptual relational scheme. There are, in general, a large choice of this transformations depending on

* the combination of attributes
* the handling of null values, and
* the handling of INDs.

Among these factors, the handling of null values and INDs on conceptual schemes depends on the logical feature of the IDM. Therefore, first we will describe the relationship between conceptual and internal schemes.

3.1 Conceptual and internal schemes in the Noah

The IDM does not directly support null values and INDs. If a tuple is inserted into a relation with some attribute values unspecified, "0" is put in as a default value when the type of the attribute in question is integer. When the type is character, the attribute field of the tuple is left blank.

Null values on the conceptual scheme of the Noah are virtually realized by the Query Processor's transformations of queries and data. We will describe the relationship between conceptual and internal schemes using an example.

Let us assume we have obtained the entity set shown in

Fig.3.1 as a part of an enterprise scheme. Since B is a partial attribute, the corresponding relation (let us call it R_E) contains null values. To cope with these null values, the following three approaches can be considered.

(a) to allow null values on the conceptual schemes

(b) not to allow null values on the conceptual schemes

(b-1) decompose R vertically

(b-2) decompose R horizontally

Each approach is illustrated in Fig.3.2. Note that an IND: $R_E[K] \supseteq R_{E'}[K]$ holds in the case of (b-1), and an IND: $R_{E_1}[K] \cap R_{E_2}[K] = \emptyset$ holds in the case of (b-2). (These INDs are maintained by the Query Processor.) In the approach (b), conceptual schemes directly correspond to internal schemes. However in the approach (a) conceptual schemes need to be transformed into internal scheme where null values are not directly supported. There are the following three variations for this transformation.

(i) Vertical decomposition

(ii) Horizontal decomposition

(iii) Introduction of a new attribute to distinguish null values

Methods (i) and (ii) are respectively based on the same idea as the approaches (b-1) and (b-2) stated above. To recover a relation on the conceptual scheme, outer-join (method (i)) and outer-union (method (ii)) are performed by the Query Processor.

Internal scheme by method (iii) which corresponds to R_E in Fig.3.2(a) is illustrated in Fig.3.3.

3.2 The design of conceptual schemes

For each entity or relationship set of an enterprise scheme, the transformation into a conceptual scheme consists of the following three parts.

* separate 1:n- and m:n-attributes to make the transformed relational schemes in 3NF.
* combine 1:1- and m:1-attributes properly.
* for partial attributes, apply one of the three methods stated in Section 3.1.

Different conceptual schemes can be obtained depending on the combination and the order of applications of above three transformations. Designers must select the conceptual scheme which is considered to be optimal for him among many variations. An example of the transformation is shown below.

[Example 3.1]

An enterprise scheme shown in Fig.2.9 can be transformed into a conceptual scheme shown in Fig.3.4 where ERD is used as a description tool of a conceptual scheme. Following the transformation rule given in Section 2.2, the ERD in Fig.3.4 becomes equivalent to the following relational schemes.

STUDENT_WITH_DEGREE (S#, SNAME, DEGREE)

STUDENT_WITHOUT_DEGREE (S#, SNAME)

ENROLL (S#, C#, SCORE)

COURSE (C#, CNAME)

TEACH (T#, C#)

TEACHER (T#, TNAME)

TEACHER_WITH_ROOM (T#, ROOM#)

INDs:

ENROLL[S=] $\subseteq$ STUDENT_WITH_DEGREE[S#]

$\cup$ STUDENT_WITHOUT_DEGREE[S#]

STUDENT_WITH_DEGREE[S#] $\cap$ STUDENT_WITHOUT_DEGREE[S#] = $\emptyset$

ENROLL[C#] = COURSE[C#]

COURSE[C#] = TEACH[C#]

TEACH[T#] $\subseteq$ TEACHER[T#]

TEACHER_WITH_ROOM[T#] $\subseteq$ TEACHER[T#]

Objects:

{S#, C#}, {S#, C#, SCORE} for ENROLL

There were three partial attributes in the enterprise scheme. In the transformation process, null values in these three attributes SCORE, ROOM# and DEGREE were coped with following the approaches (a), (b-1) and (b-2), respectively.

4. The maintenance of semantic constraints by the Query Processor

In this chapter, we will show how the Query Processor

virtually realizes the conceptual schemes supported with high-level semantic constraints such as INDs, null values and object patterns.

(I) INDs

To maintain INDs, the Query Processor need to check whether update operations issued by users are violating the INDs or not. For example, let us consider an IND: TEACH[T#] $\subseteq$ TEACHER[T#] in Example 3.1. The following two kinds of update commands need to be checked for the maintenance of the IND.

(i) an insertion of tuple(s) into the relation TEACH

(ii) a deletion of tuple(s) from the relation TEACHER

For (i), if a command

```
append to teach(t# = '3', c# = '310')                    (4-1)
```

is issued by a user, the Query Processor need to transform it into the following commands.

```
range of tr is teacher
append to teach (t# = '3', c# = '310')                   (4-2)
where count(tr.t# where tr.t# = '3') > 0
```

Furthermore, if the response of the IDM to the command (4-2) is "0 tuples affected", the Query Processor should inform a user that the command (4-1) caused no effect to the current instance of the database.

Next in the case of (ii), for example, if a user issued the following command (4-3), the Query Processor transforms it into the one in (4-4).

```
range of tr is teacher
delete tr where tr.t# = '5'                                    (4-3)
```

```
range of tr is teacher
range of t is teach
delete tr where tr.t# = '5'                                    (4-4)
and count(t.t# where t.t# = '5') = 0
```

In this case if the IDM replied as "0 tuples affected", the Query Processor either informs the user of this fact or deletes a tuple of which T#-value is '5' from the relation TEACH.

(II) Null values

As shown in Section 3.1, there are three methods to cope with null values in conceptual schemes. Consider the attribute SCORE in Fig.3.4, and assume that method (iii) of Section 3.1 is used. Although the relational scheme having the attribute SCORE is ENROLL(S#, C#, SCORE), the corresponding internal scheme needs one more attribute, say IS_SCORE, to explicitly represent the existence of values in SCORE. For example, "0" and "1" are used to imply that the SCORE-value of the tuple is null and non-null, respectively. (See Fig.3.3.) The detail of the technique is given in [BLI 8204].

(III) Objects

Let us consider the running example in Example 3.1 again. Object sets {S#, C#} and {S#, C#, SCORE} are posed for the relation ENROLL. Therefore, if a user tries to insert a tuple of which S#-value is not specified, the Query Processor should reject the insertion request. Similarly all the attempts of updates such that the resultant relation violates the object sets restriction should be rejected.

5. Conclusion

We have described the design principle of conceptual/internal schemes of the Noah, and proposed the enhancement of logical power of the IDM by the Query Processor. For the implementation of this enhancement, several problems must be solved which include the handling of data dictionary (called system relation in the IDM.)

Acknowledgments

The authors would like to express their sincere appreciations to Professor Yajima and Associate Professor Kambayashi for their continuous encouragement. The authors also wish to thank to the members of Yajima Laboratory for their fruitful discussions.

References

[BLI 8109] Britton-Lee, Inc., "IDM 500 Software Reference Manual", Version 1.3, Sept. 1981.

[BLI 8204] Britton-Lee, Inc., "NULL DATA VALUES on the IDM", Britton-Lee, Inc., Technical Bulletin, No.TB-001, pp.3-5, Apr.1982.

[CHEN 7603] Chen,P.P., "The Entity-Relationship Model - Toward a Unified View of Data", ACM Trans. on Database Systems, Vol.1, No.1, pp.9-36, Mar. 1976.

[CHEN 7706] Chen,P.P., "The Entity-Relationship Model - A Basis for the Enterprise View of Data", Proc. of AFIPS NCC, pp.77-84. June 1977.

[KAMBK8302] Kambayashi,Y., Kojima,I., Yazaki,T. and Yajima,S., "A Micro-Computer-Based Relational Database System with Database Preparation Facilities", Proc. of the Conference on Relational DBMS Design/Implementation/Use in a Micro-Computer Environment, Feb. 1983.

[LEVI 8302] Le Viet,C., "The Noah Database Machine", Proc. of IEEE COMPCON, pp.364-368. Feb.-Mar. 1983.

[LEVI 83] Le Viet,C., "Noah - A Relational Database System Based on the Intelligent Database Machine", to appear in a special issue of "bit".

[SMITS7706] Smith,J.M. and Smith,D.C.P., "Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol.2, No.2, pp.105-133, June 1977.

[WONGK8209] Wong,H.K.T. and Kuo,I., "GUIDE: Graphical User Interface for Database Exploration", Proc. of 8th International Conference on VLDB, pp.22-31, Sept. 1982.

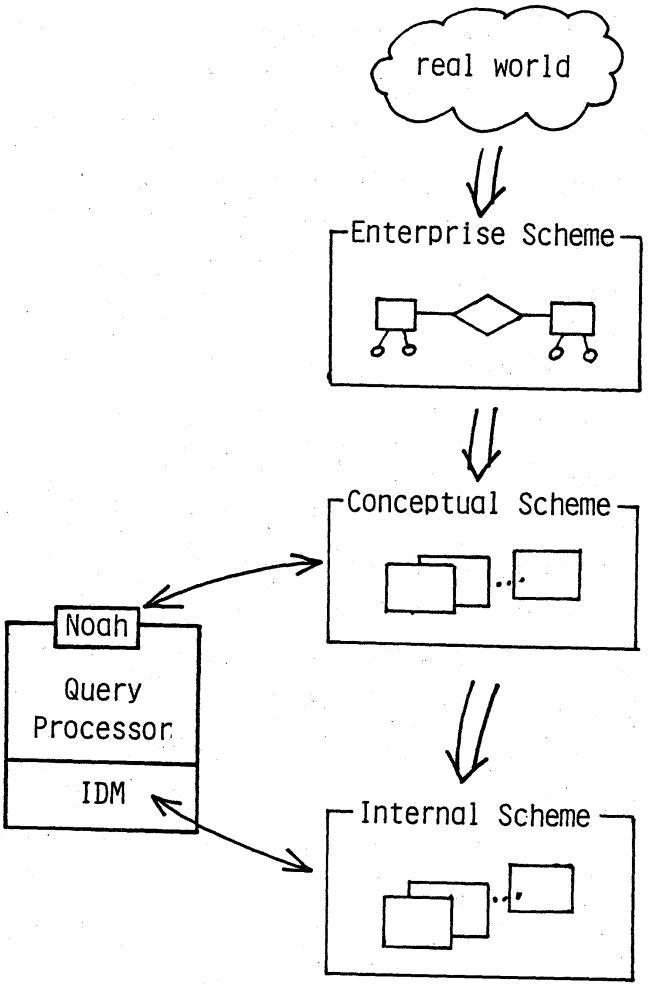

Fig.1.1 The configuration of the design process.

E:A mapping	1:1	m:1	m:n	1:n
total	E ↔ A	E → A	E — A	E ← A
partial	E ⇢ A	E ⇢ A	E ⋯ A	E ⇠ A

Table 2.1 Types of edges between an entity set and an its attribute.

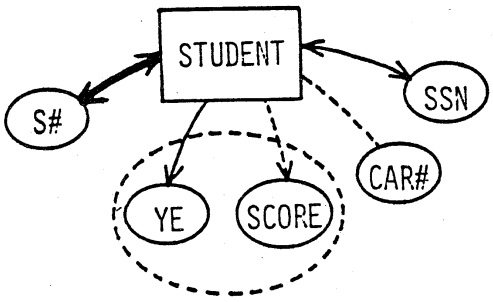

Fig.2.1 An entity set and its attributes.

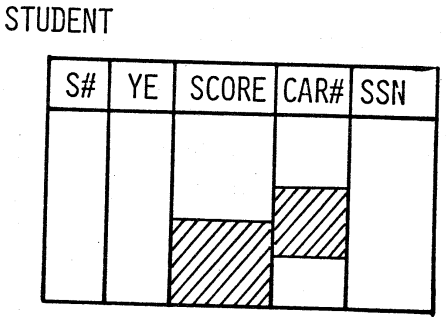

Fig.2.2 A typical object pattern of the relation STUDENT.

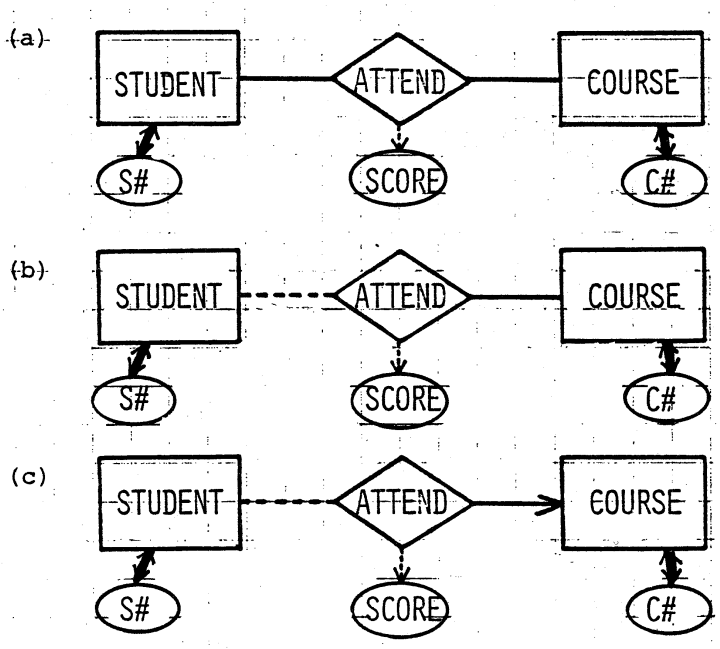

Fig.2.3 Some types of relationship sets.

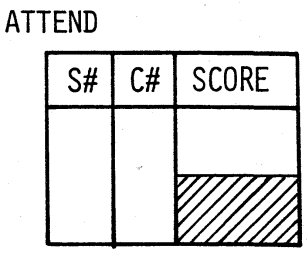

Fig.2.4 A typical object pattern of the relation ATTEND.

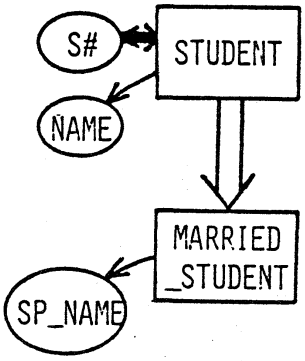

Fig.2.5 A sub-entity set.

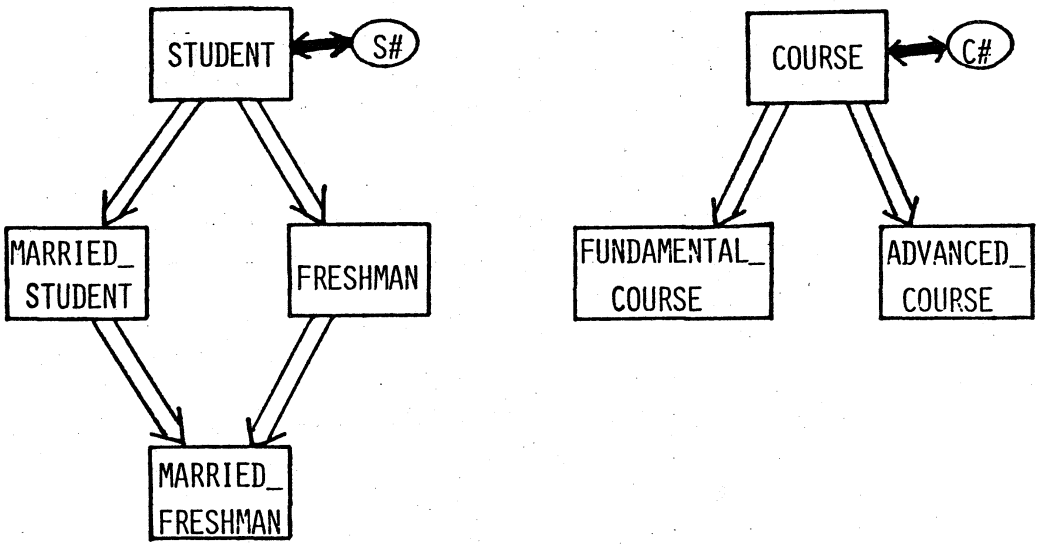

Fig.2.6 Sub-entity structures.

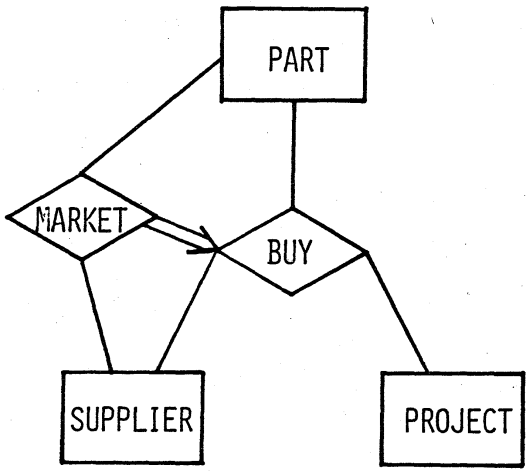

Fig.2.7 A sub-relationship set.

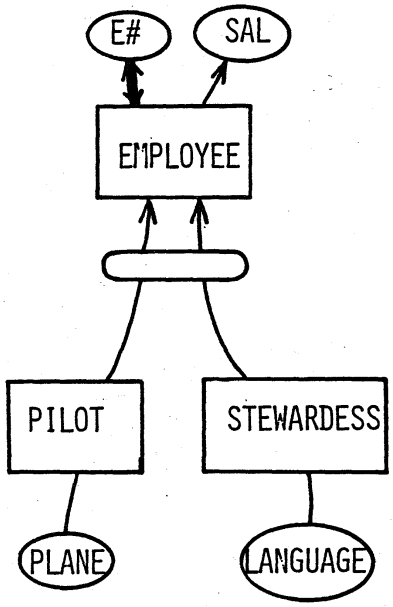

Fig.2.8 Mutually exclusive entity sets.

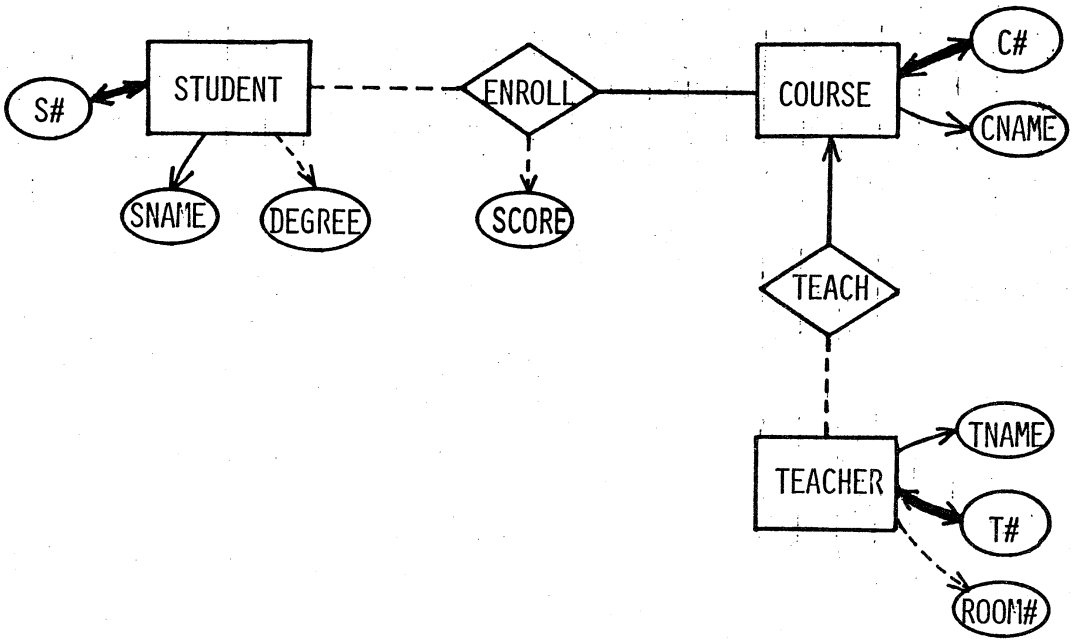

Fig.2.9 An enterprise scheme.

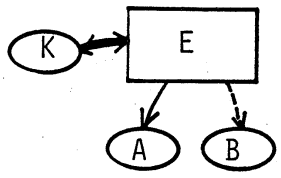

Fig.3.1 An entity set with
a partial attribute.

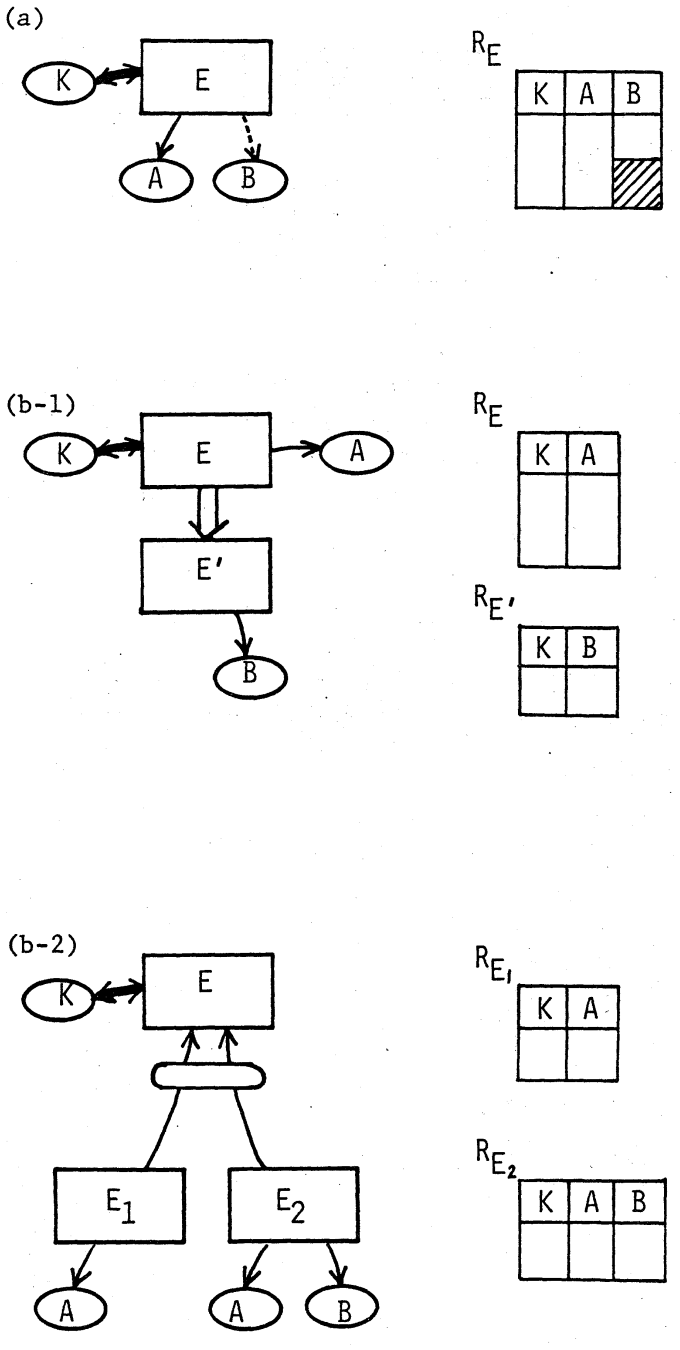

Fig.3.2 Three approaches to cope with null values.

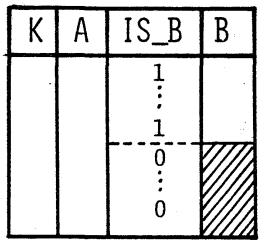

Fig.3.3 Introduction of a new attribute which ditinguish null values.

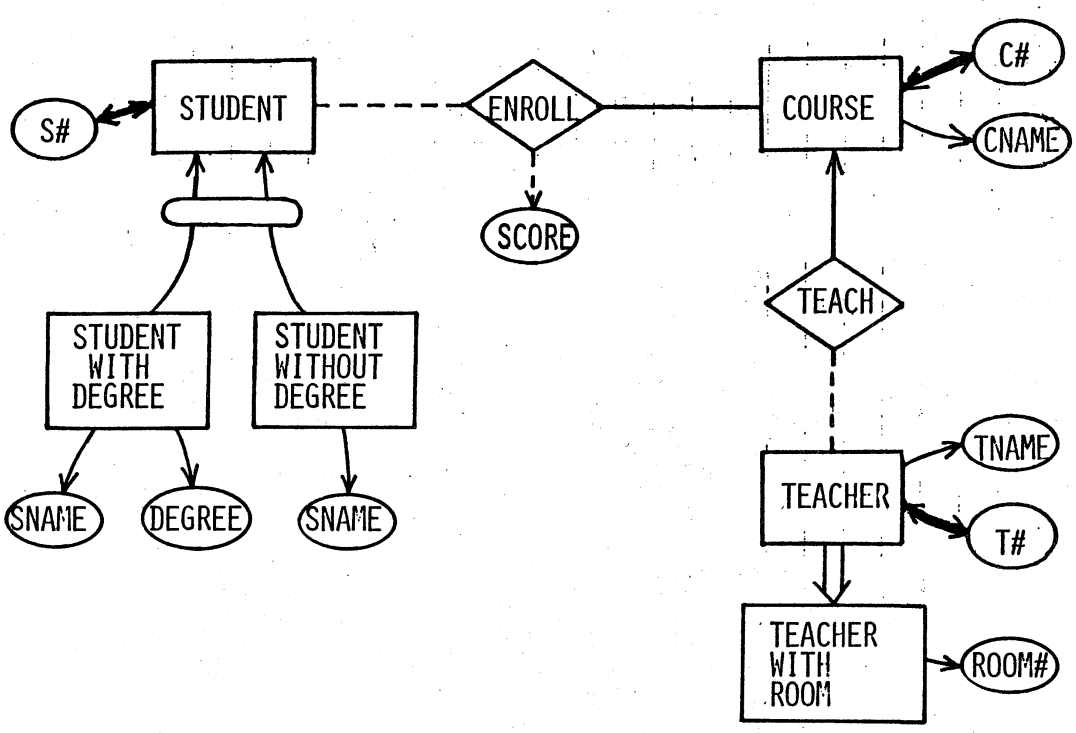

Fig.3.4 A conceptual scheme.